Simple voting algorithms for Italian parsing

Simple Voting Algorithms for Italian Parsing

Alessandro Mazzei

Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 101049 Torino, Italy
mazzei@di.unito.it

Abstract. This paper presents an ensemble system for dependency parsing of Italian: three parsers are separately trained and combined by means of a majority vote. The three parsers are the MATE parser[1], the DeSR parser[2], and the MALT parser[3]. We present three experiments showing that a simple voting combination further improves the performances of the parsers.

1 Introduction

In the last few years Natural Processing Language (NLP) community devoted great attention to the dependency formalisms and many practical NLP systems adopted the dependency parsing [16]. Larger dependency treebanks and more sophisticated parsing algorithms improved the performances of dependency parsers for many languages [21, 13]. For instance, dependency parsing for Italian constantly increased its performances. As reported in the Evalita evaluation campaigns [12], the best scores for Italian dependency parsing (expressed in Labelled Attachment Score, LAS) was 86.94% in 2007, 88.73% in 2009, and 91.23% in 2011 [8]. These results have been obtained by using the Turin University Treebank, a dependency treebank for Italian [7] (see Section 3). However, statistical dependency parsing seems to still have room for improving. On the one hand, new promising specific algorithms for learning and classification are emerging; on the other hand, universal machine learning techniques seem to be useful for this specific task. Some algorithms use larger sets of syntactic features (e.g. [19, 10]), while others are trying to apply general techniques *to combine* together the results of various parsers [26, 23, 14, 3, 24, 17]. We designed three experiments on parser combination for Italian that follows both these directions.

We employed three state of the art statistical parsers, which use sophisticated parsing algorithms and advanced feature sets. The three parsers are the MATE parser [6], the DeSR parser [2], the MALT parser [22]. Moreover, in our system we combined these three parsers by using two very simple voting algorithms [9, 26]. We decided to apply an "out of box" approach, i.e. we applied each parser with its standard configurations for learning and classification.

[1] http://code.google.com/p/mate-tools/, version 2.0
[2] http://sites.google.com/site/desrparser/
[3] http://maltparser.org/

Now we give a brief description of the three parsers applied in our experiments, i.e. MATE, DeSR and MALT parsers.

The MATE parser [5, 6] is a development of the algorithms described in [10, 15]. It basically adopts the second order maximum spanning tree dependency parsing algorithm. In particular, Bohnet exploits *hash kernel*, a new parallel parsing and feature extraction algorithm that improves the accuracy as well as the parsing speed [6]. The MATE performances on English and German, which are 90.14% and 87.64% respectively (LAS), posed this parser at the state of the art for these languages [13, 6, 1].

The DeSR parser [2] is a transition (shift-reduce) dependency parser similar to [25]. It builds dependency structures by scanning input sentences in left-to-right and/or right-to-left direction. For each step, the parser learns from the annotated dependencies if to perform a shift or to create a dependency between two adjacent tokens. DeSR can use different set of rules and includes additional rules to handle non-projective dependencies. The parser can choose among several learning algorithms (e.g Multi Layer Perceptron, Simple Vector Machine), providing user-defined feature models. In our experiments we adopted for DeSR the Multi Layer Perceptron algorithm, which is the same configuration that the parser exploited when it won the Evalita 2009 competition.

The MALT parser [22] implements the transition-based approach to dependency parsing too. In particular MALT has two components: (1) a (non-deterministic) transition system that maps sentences to dependency trees; (2) a classifier that predicts the next transition for every possible system configuration. MALT performs a greedy deterministic search into the transition system guided by the classifier. In this way, it is possible to perform parsing in linear time for projective dependency trees and quadratic time for arbitrary (non-projective) trees [20]. MALT has several built-in transition systems, but in our experiments we adopted just the standard "Nivre arc-eager" system, that builds structure incrementally from left to right. Moreover, we use the standard classifier provided by MALT, i.e. the SVM (Simple Vector Machine) basic classifier on the standard "NivreEager" feature model.

To our knowledge this is the first work that experimented the MATE parser on Italian, while DeSR and MALT parsers have been used in many occasions on Italian (e.g. [17, 4]), reaching the best results in several contests. In the next Sections we describe our approach for ensemble parsing (Section 2) and we report the results of three experiments (Section 3), before concluding the paper (Section 4).

2 The combination algorithms

In order to combine the three parsers we used two very simple algorithms, COM1 and COM2 (see algorithms 1 and 2), both implemented in the PERL programming language. These algorithms have been previously experimented in [26] and in [24]. The main idea of the COM1 algorithm is to do a democratic voting

among the parsers. For each word[4] of the sentence, the dependency (the parent and the edge label) assigned to the word by each parser is compared: if at least two parsers assign the same dependency, the COM1 algorithm selects that dependency. In the case that each parser assigns a different dependency to the word, the algorithm selects the dependency assigned by the "best parser". As noted by [26], who use the name *voting* for COM1, this is the most logical decision if it is possible to identify a priori the "best parser", in contrast to the more democratic random choice.

```
foreach sentence do
    foreach word in the sentence do
        if DependencyParser2(word) == DependencyParser3(word) then
            DependencyParser-COM1(word) := DependencyParser2(word)
        else
            DependencyParser-COM1(word) := DependencyParser1(word)
        end
    end
end
```

Algorithm 1: The combination algorithm COM1, that corresponds to the *voting* algorithm reported in [26]

The COM2 algorithm is a variation of the COM1. COM1 is a single word combination algorithm that does not consider the whole dependency structure. This means that incorrect dependency trees can be produced by the COM1 algorithm: cycles and multiple roots can *corrupt* the "treeness" of the structure. The solution that we adopt in the COM2 algorithm is naive: if the tree produced by the COM1 algorithm for a sentence is corrupted, then the COM2 returns the tree produced by the "best parser". Again, similarly to [26], who use the name *switching* for COM2, this is the most logical decision when there is an emerging best parser from a development data set.

[4] In this paper we use the term *word* in a general sense, as synonym of *token*.

```
foreach sentence do
    foreach word in the sentence do
        if DependencyParser2(word) == DependencyParser3(word) then
            DependencyParser-COM2(word) := DependencyParser2(word)
        else
            DependencyParser-COM2(word) := DependencyParser1(word)
        end
    end
    if TREE-COM2(sentence) is corrupted then
        TREE-COM2(sentence) := TREE-PARSER1(sentence)
    end
end
```

Algorithm 2: The combination algorithm COM2, that corresponds to the *switching* algorithm reported in [26]

3 Experimental Results

We applied our approach for parsing combination in three experiments. In the first experiment we use the datasets provided in the SPLeT competition [11], in the second experiment we used the datasets provided in the Evalita 2011 competition [8], and in the third experiment we used both the datasets. For all the experiments we used two machines. A powerful Linux workstation, equipped with 16 cores, processors 2GHz, and 128 GB ram has been used for the training of the MATE parser, that is the most computationally expensive system: on this machine the average training time for MATE was 8 hours.

Another Linux workstation equipped with a single processor 1GHz, and 2 GB ram has been used for the training of the DeSR and MALT parsers: that usually required a couple of hours. This machine has been used for testing all the systems: this phase required several minutes for MATE parser and few minutes for MALT and DeSR parsers. MALT and DeSR parsers accept as input the CONLL-07 format, that is the format provided by the SPLeT organizers. In contrast, MATE accepts the CONLL-09 format: simple conversions scripts have been implemented to manage this difference.

3.1 The SPLeT experiment

In the SPLeT experiment, a first run was performed in order to evaluate the "best parser" in the COM1 and COM2 algorithms. We used the ISST training[5] (71,568 words, 3,275 sentences) as training set and the ISST development[6] (5,165 words, 231 sentences) as development set. The first row in Table 1 shows the results of the three parsers in this first experiment. MATE parser outperforms the DeSR and MALT parsers: MATE does $\sim$3% better than DeSR and $\sim$5% better than

[5] `File:` *it_isst_train.splet*
[6] `File:` *it_isst_test.splet*

	MATE	**DeSR**	**MALT**	**COM1**	**COM2**	**BL**$_{W_2}$	**BL**$_{W_3}$	**BL**$_{W_4}$
DevSet	81.92	78.99	77.04	82.54	82.36	81.45	82.54	82.63
TestSet	82.57	78.68	77.98	83.20	**83.08**	82.23	83.15	83.24
NatReg	75.76	70.66	70.33	76.28	75.88	74.78	76.07	75.97

Table 1. The performances (LAS score) of the three parsers, their simple combination (COM1 and COM2), their blended combination (Blended$_{W_2}$, Blended$_{W_3}$, Blended$_{W_4}$) on the SPLeT test set, development set, Regional laws set.

MALT. On the basis of this result, we used MATE as our "best parser" in the combination algorithms (cf. Section 2). COM1 and COM2 reach the score of 82.54% and 82.36% respectively, and so both combination algorithms improve the performances of the MATE parser close to the 0.5%.

In a second run, we used the whole ISST as training set[7] (total 76,733 words, 3,506 sentences) and we used the blind file provided by the organizers as test set[8] (5,662 words, 240 sentences, European Directives Laws). The second row in Table 1 shows the results of the three parsers in this second experiment: the value 83.08%, produced by the COM2 algorithm, is the final result of our participation to the SPLeT shared task [18]. Note that there is a $\sim$0.1% difference between the COM1 and COM2 results: similar to [26, 24] we have 10 corrupted trees in the test set, i.e. $\sim$ 4% of the total (240 sentences). In Table 2 we detailed the results of the three parsers in the SPLeT experiment on the basis of their agreement. When the three parsers agree on the same dependency (Table 2, first row), this happens on $\sim$ 72% of the words, they have a very high LAS score, i.e. 95.6%. Moreover, DeSR and MALT parsers do better than the MATE parser only when they agree on the same dependency (Table 2, second row). The inspection of the other rows in Table 2 shows that COM1 algorithms has the best possible performance w.r.t. the voting strategy. In other words, COM1 selects all the parser combinations that correspond to higher value of LAS score (cf. the discussion on *minority dependencies* in [24]).

In a third run, we again use the whole ISST as training set[9] (total 76,733 words, 3,506 sentences), but we use the NatReg file provided by the organizers as test set[10] (5,194 words, 119 sentences, Regional Laws of Piedmont Region). The third row in Table 1 shows the results of the three parsers in this third run: in this case we have 75.88% for COM2 algorithm. This lower result can be advocated to the different nature of the domain. It is interesting to note that in this experiment MALT and DeSR parsers give similar results ($\sim$70%), while the MATE parser still outperforms them by $\sim$5%.

[7] `File:` *it_isst_train.splet* `and` *it_isst_test.splet*
[8] `File:` *it_EULaw_test_blind.splet*
[9] `File:` *it_isst_train.splet* and *it_isst_test.splet*
[10] `File:` *it_NatRegLaw_test_blind.splet*

Scores			Frequency
MATE == **95.6**	DeSR ==	MALT	**71.99**
MATE != 30.7	**DeSR** == **45.8**	**MALT**	**4.20**
MATE ==	**DeSR** != **67.2**	MALT 14.4	**7.70**
MATE ==	**MALT** != **59.1**	DeSR 20.0	**8.21**
MATE != **31.1**	DeSR != 14.5	MALT 16.3	**7.89**

Table 2. The detailed performances (LAS score) of the three parsers and their simple combination on the SPLeT blind set, corresponding to the first row of the Table 1.

3.2 The EVALITA experiment

We performed a second experiment on two different training and test sets belonging to a different Italian Treebank, which has a different PoS tag set and a different dependency label set. We used for learning the Evalita 2011 Development Set[11] (93,987 words, 3,452 sentences; balanced corpus of newspapers, laws, wikipedia) and we use for testing the Evalita 2011 test set[12] (7,836 words, 300 sentences; balanced corpus): these sets have been annotated according to the format of the Turin University Treebank [8]. The first row in Table 3 shows the results of the three parsers in this experiment: in this case we have 89.16% for COM2. It is interesting to note that the improvement of the COM2 algorithm with respect to the MATE parser is only $\sim 0.1\%$. In Table 4 we detailed the results of the three parsers in this run on the basis of their agreement. Again, when the three parsers agree on the same dependency (Table 4, first row) which happens for $\sim 78\%$ of the words, they have a very high LAS score, i.e. 96.6%. In contrast with the SPLeT experiment, here we do not have a relevant improvement when DeSR and MALT parsers do better than the MATE parser, i.e. only when they agree on the same dependency (Table 4, second row). In other words, on the SPLeT test set, the COM1[13] algorithm does much better than MATE since DeSR and MALT parsers have a good performance (45.8% vs. 30.7%) when they do not agree with the MATE parser: this is not true for the EVALITA experiment, where DeSR and MALT have 38.8% while MATE has 35.2%.

In order to evaluate the COM1 and COM2 algorithms in a more general context, we performed two new runs on the EVALITA dataset by using other parsers. We used five parsers that have participated to the Evalita 2011 compe-

[11] `File:` *evalita*2011_*train.conll*

[12] `File:` *evalita*2011_*test.conll*

[13] The same consideration hold for COM2: in the second experiment there are just 8 corrupted trees

tition [8]. In the first run (second row in Table 3 and Table 5) we combined the Parsit[14], the UniPi[15] and the FBKirst[16] parsers, i.e. the best scored systems in the competition. In the second run (third row in Table 3 and Table 6) we combined the Parsit, UniPi and the UniTo parsers, i.e. two statistical parsers and one rule-based parser. From Table 3 we can note that the best result (92.54%) is obtained by the COM1 in the second run, i.e. when the UniTo parser belongs to the ensemble. Comparing the second rows in Table 5 and in the Table 6 we can explain this result. There is a relevant improvement when UniPi and UniTo parsers do better than the Parsit parser, i.e. the COM1 algorithm do much better than Parsit since UniPi and UniTo parsers have a good performance (29.6% vs. 58.3%) when they do not agree with the Parsit parser. This result confirms that the performance of the parsing combination depends on the "diversity" of the parsers involved rather than on the absolute score of each single parser.

MATE	**DeSR**	**MALT**	**COM1**	**COM2**	**BL**$_{W_2}$	**BL**$_{W_3}$	**BL**$_{W_4}$
89.07	86.26	80.76	89.19	89.16	88.03	89.19	89.19
Parsit	**UniPi**	**FBKirst**	**COM1**	**COM2**	**BL**$_{W_2}$	**BL**$_{W_3}$	**BL**$_{W_4}$
91.23	89.88	88.62	91.95	92.04	91.12	91.97	91.93
Parsit	**UniPi**	**UniTo**	**COM1**	**COM2**	**BL**$_{W_2}$	**BL**$_{W_3}$	**BL**$_{W_4}$
91.23	89.88	85.34	92.54	92.50	91.39	92.57	92.65

Table 3. The performances (LAS score) of the MATE, DeSR, MALT, Parsit, UniPi, FBKirst and UniTo parsers, their simple combination (COM1 and COM2), their blended combination (BLended$_{W_2}$, BLended$_{W_3}$, BLended$_{W_4}$) on the Evalita 2011 test.

3.3 Parsing combination versus Re–parsing experiment

Similar to [26], we designed the COM2 algorithm since COM1 can produce corrupted dependency trees. COM2 tests the correctness of the tree and in the case of corruption returns the dependency structure produced by the "best parser" of the ensemble. We hypothesized that this strategy can produce good results in our system since one of the parser of the ensemble drastically outperforms the others. However, some more general solution to the tree-corruption problem have been proposed: the re–parsing strategy [23, 14, 3]. In re–parsing, a new, not corrupted, dependency tree is produced by taking into account the trees produced by each parser of the ensemble. Attardi and Dell'Orletta proposed an approximate top-down algorithm that starts by selecting the highest-scoring root node, then the highest-scoring children and so on [3]. Sagae and Lavie together with Hall et al. proposed a two-steps algorithm: (1) to create a graph by merging all

[14] `http://www.parsit.it`
[15] The UniPi parser is the DeSR parser tuned for this specific competition.
[16] The FBKirst parser is an ensemble combination of the MALT parser.

Scores			Frequency
MATE == **96.6**	DeSR ==	MALT	**78.39**
MATE != 35.2	**DeSR** == **38.8**	**MALT**	**3.38**
MATE == **82.0**	**DeSR** !=	MALT 7.2	**9.17**
MATE == **63.3**	**MALT** !=	DeSR 19.6	**4.27**
MATE != **40.7**	DeSR != 18.4	MALT 7.9	**4.78**

Table 4. The detailed performances (LAS score) of the MATE, DeSR and MALT parsers and their combination on the Evalita 2011 test set.

Scores			Frequency
Parsit == **97.7**	UniPi ==	FBKirst	**85.15**
Parsit != 37.7	**UniPi** == **49.0**	**FBKirst**	**6.34**
Parsit == **75.9**	**UniPi** !=	FBKirst 9.4	**3.59**
Parsit == **66.8**	**UniPi** !=	FBKirst 19.5	**2.57**
Parsit != **52.3**	UniPi != 16.1	FBKirst 12.6	**7.89**

Table 5. The detailed performances (LAS score) of the Parsit, UniPi and FBKirst parsers on the Evalita 2011 test set.

the structures produced by the parser on the ensemble, and (2) to extract the most probable dependency spanning tree from this graph [23, 14].

Surdeanu and Manning provided experimental evidence that re–parsing algorithms are a good choice for practical ensemble parsing in out domains [24]: in order to confirm this hypothesis we performed a third experiment on both the SPLeT and EVALITA datasets by using the "MaltBlender" tool [14]. In Table 1 and Table 3 the columns $\mathbf{BL}_{W_2}$, $\mathbf{BL}_{W_3}$, $\mathbf{BL}_{W_4}$ report the application of the algorithm described in [14]. There are three weighting strategies: the results of the three parsers are equally weighted (W_2); the three parsers are weighted according to the total labeled accuracy on a held-out development set (W_3); the parsers are weighted according to labeled accuracy per coarse grained PoS tag on a held-out development set (W_4). For the first, the second and the third runs of the SPLeT experiment (Table 1), the held-out development set is the SPLeT development set; for the EVALITA experiment (Table 3), the held-out development set is the Evalita 2011 test set.

Scores			Frequency
Parsit == UniPi == UniTo **98.2**			**80.92**
Parsit != **UniPi** == **UniTo** 29.6	**58.3**		**4.57**
Parsit == **UniPi** != UniTo 	**81.7**	9.3	**7.82**
Parsit == **UniPi** != UniTo 	**72.1**	15.5	**2.96**
Parsit != UniPi != UniTo **49.6**	23.2	8.3	**3.73**

Table 6. The detailed performances (LAS score) of the Parsit, UniPi and UniTo parsers on the Evalita 2011 test set.

Three evidences seems to emerge from the third experiment: (1) the re–parsing strategy always performs slightly better than COM2 algorithm but not always better than COM1 algorithm; (2) there is no winning weighting strategy for re–parsing; (3) it does not seem that blending performs better out domain than in domain.

4 Conclusions

In this paper we described three parsing experiments on three parsers, i.e. the MATE, the DeSR and the MALT parsers. The first emerging issue by these experiments is that the MATE parser has a very good performance on Italian ISST treebank, both in domain and out domain, reaching very good scores. The EVALITA experiment confirms that similar results can be obtained on the Turin University Treebank. The second emerging issue is that very simple combination algorithms, as well as more complex blending algorithms, can furthermore improve performance also in situations where one parser outperforms the others.

In future research we plan to repeat our experiments on a larger set of parsers. In particular, on the basis of the results emerged by the EVALITA experiment, i.e. that "diversity" is an important value in combining parsers, we want to perform more tests on the combination of statistical parsers with rule based parsers.

References

1. Anders, B., Bernd, B., Hafdell, L., Nugues, P.: A high-performance syntactic and semantic dependency parser. In: Coling 2010: Demonstrations. pp. 33–36. Coling 2010 Organizing Committee, Beijing, China (August 2010), `http://www.aclweb.org/anthology/C10-3009`
2. Attardi, G.: Experiments with a multilanguage non-projective dependency parser. In: Proceedings of the Tenth Conference on Computational Natural Language

Learning (CoNLL-X). pp. 166–170. Association for Computational Linguistics, New York City (June 2006), http://www.aclweb.org/anthology/W/W06/W06-2922
3. Attardi, G., dell'Orletta, F.: Reverse revision and linear tree combination for dependency parsing. In: HLT-NAACL. pp. 261–264 (2009)
4. Attardi, G., Simi, M., Zanelli, A.: Tuning DeSR for the Evalita 2011 Dependency Parsing. In: Working Notes of EVALITA 2011. CELCT a r.l. (2012), iSSN 2240-5186
5. Bohnet, B.: Efficient parsing of syntactic and semantic dependency structures. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task. pp. 67–72. CoNLL '09, Association for Computational Linguistics, Stroudsburg, PA, USA (2009), http://dl.acm.org/citation.cfm?id=1596409.1596421
6. Bohnet, B.: Top accuracy and fast dependency parsing is not a contradiction. In: Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010). pp. 89–97. Coling 2010 Organizing Committee, Beijing, China (August 2010), http://www.aclweb.org/anthology/C10-1011
7. Bosco, C., Lombardo, V.: Dependency and relational structure in treebank annotation. In: Proceedings of the COLING'04 workshop on Recent Advances in Dependency Grammar. Geneve, Switzerland (2004), http://www.di.unito.it/~bosco/publicat/dependency-coling04.zip
8. Bosco, C., Mazzei, A.: The evalita 2011 parsing task: the dependency track. In: Working Notes of EVALITA 2011. CELCT a r.l. (2012), iSSN 2240-5186
9. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
10. Carreras, X.: Experiments with a higher-order projective dependency parser. In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007. pp. 957–961 (2007), http://www.aclweb.org/anthology/D/D07/D07-1101
11. Dell'Orletta, F., Marchi, S., Montemagni, S., Plank, B., Venturi, G.: The SPLeT-2012 Shared Task on Dependency Parsing of Legal Texts. In: SPLeT 2012 – Fourth Workshop on Semantic Processing of Legal Texts (SPLeT 2012) – First Shared Task on Dependency Parsing of Legal Texts (2012)
12. EVALITA 2011 Organization Comitee: Working Notes of EVALITA 2011. CELCT a r.l (2012)
13. Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M.A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., Zhang, Y.: The conll-2009 shared task: syntactic and semantic dependencies in multiple languages. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task. pp. 1–18. CoNLL '09, Association for Computational Linguistics, Stroudsburg, PA, USA (2009), http://dl.acm.org/citation.cfm?id=1596409.1596411
14. Hall, J., Nilsson, J., Nivre, J., Eryigit, G., Megyesi, B., Nilsson, M., Saers, M.: Single malt or blended? a study in multilingual parser optimization. In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007. pp. 933–939 (2007), http://www.aclweb.org/anthology/D/D07/D07-1097
15. Johansson, R., Nugues, P.: Dependency-based syntactic-semantic analysis with propbank and nombank. In: Proceedings of the Twelfth Conference on Computational Natural Language Learning. pp. 183–187. CoNLL '08, Association for Computational Linguistics, Stroudsburg, PA, USA (2008), http://dl.acm.org/citation.cfm?id=1596324.1596355
16. Kübler, S., McDonald, R.T., Nivre, J.: Dependency Parsing. Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers (2009)

17. Lavelli, A.: An Ensemble Model for the EVALITA 2011 Dependency Parsing Task. In: Working Notes of EVALITA 2011. CELCT a r.l. (2012), iSSN 2240-5186
18. Mazzei, A., Bosco, C.: Simple Parser Combination. In: SPLeT 2012 – Fourth Workshop on Semantic Processing of Legal Texts (SPLeT 2012) – First Shared Task on Dependency Parsing of Legal Texts. pp. 57–61 (2012)
19. McDonald, R., Pereira, F.: Online learning of approximate dependency parsing algorithms. In: Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)). vol. 6, pp. 81–88 (2006)
20. Nivre, J.: Algorithms for deterministic incremental dependency parsing. Computional Linguistics 34(4), 513–553 (Dec 2008)
21. Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., Yuret, D.: The CoNLL 2007 shared task on dependency parsing. In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007. pp. 915–932 (2007), http://www.aclweb.org/anthology/D/D07/D07-1096
22. Nivre, J., Hall, J., Nilsson, J.: Maltparser: a data-driven parser-generator for dependency parsing. In: Proceedings of LREC-2006. vol. 2216-2219 (2006)
23. Sagae, K., Lavie, A.: Parser combination by reparsing. In: Moore, R.C., Bilmes, J.A., Chu-Carroll, J., Sanderson, M. (eds.) HLT-NAACL. The Association for Computational Linguistics (2006)
24. Surdeanu, M., Manning, D.C.: Ensemble models for dependency parsing: Cheap and good? In: NAACL. The Association for Computational Linguistics (2010)
25. Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In: Proceedings of IWPT. vol. 3 (2003)
26. Zeman, D., Žabokrtskỳ, Z.: Improving parsing accuracy by combining diverse dependency parsers. In: International Workshop on Parsing Technologies. Vancouver, Canada. pp. 171–178. Association for Computational Linguistics (2005)